INTRODUCTION TO COMPUTER SCIENCE I
PROJECT 3
Sudoku

1 The game of Sudoku

Sudoku is puzzle/game well suited for computers because it is a matter of finding legal placements for numbers on the board; it is an example of a *constraint satisfaction problem*. Sudoku is typically played on a 9×9 grid. Initially, a few of the cells of that grid contain a numbers, all integers between 1 and 9. The goal for the player is to fill in the remaining cells of the grid, also with integers between 1 and 9, given the following constraints:

- Each value (1 through 9) must appear exactly once in each row.
- Each value (1 through 9) must appear exactly once in each column.
- Each value (1 through 9) must appear exactly once in each 3×3 *sub-grid*, where the 9×9 board is divided into 9 such sub-grids.

A *valid* Sudoku game begins with initial numbers that, when combined with the constraints above, admit to **exactly one** complete solution. That is, the initial numbers cannot make it impossible to fill in the board legal, nor can they allow multiple solutions.

2 Your assignment

Getting started: Create a new folder/directory for `project-3`, and open/change into it. Then go to the following link for the starting code:

`bit.ly/COSC-111-project-3-source`

The file you will download is a *ZIP archive*; you should be able to open it and extract the files into your `project-3` directory. Specifically, the archive contains:

- `Sudoku.java`: The usual source code file that sets up the assignment and in which you will write your code.
- `Support.class`: A pre-compiled collection of methods that you will use from `Sudoku.java` to test and verify your solutions.
- Three `.board` files, described below, that contain pre-made Sudoku puzzles.

The overall goal: You need to complete `Sudoku.java` such that it can be run like this:

```
$ java Sudoku medium.board
Initial board:
0 0 0 0 0 0 0 0 7
0 0 0 0 7 0 3 0 8
0 0 0 5 0 4 0 6 0
0 0 0 0 0 0 0 8 0
7 1 0 0 9 0 0 0 5
8 0 0 0 1 5 9 0 0
3 0 0 0 0 0 0 0 0
0 0 8 9 4 0 6 3 0
0 2 7 6 0 3 0 0 0
Final board:
6 8 1 2 3 9 5 4 7
2 4 5 1 7 6 3 9 8
9 7 3 5 8 4 2 6 1
5 9 4 7 6 2 1 8 3
7 1 6 3 9 8 4 2 5
8 3 2 4 1 5 9 7 6
3 6 9 8 2 1 7 5 4
1 5 8 9 4 7 6 3 2
4 2 7 6 5 3 8 1 9
Correct solution!
```

That is, the file `medium.board` contains a Sudoku puzzle (with `0` values where there would normally be blank entries). Go ahead, open it in *Emacs* and look. Your `Sudoku` program must read that board into a two-dimensional array of `int`. It must then solve the puzzle.

Using the `Support` methods: There are two methods in the `Support` class that you must use in your program. To use each, you call them from your `Sudoku` program by using the prefix `Support.` (including the dot!). That is, to call, for example, `printResults()` (documented below), then you would write the call as `Support.printResults(isSolved, board)`.

- `public static void printAndVerifyInitialBoard (int[][] board, String initialStatePath)`

  When your program has read the initial state of the board from its file into a two-dimensional array of `int`, it must call this method, passing **both** the 2D array **and** the file name from which it was created. This method will print the initial board; it will also check the 2D array against the file, verifying that the array is correct.

- `public static void printResults (boolean isSolved, int[][] board)`

  After your program has (tried) to solve the puzzle, it must call this method. The first parameter, `isSolved`, should indicate whether your solver believed that it solved the puzzle. The second parameter, `board`, should be the 2D array containing the solved puzzle. This method will print the 2D array that is passed. Moreover, if `isSolved` is `true`, then this method will verify that the puzzle is indeed solved. If `isSolved` is `false`, this method will determine whether a solution should have been found.

A suggested approach: Do not try to write all of this code at one. Break it into pieces, program and test each piece, and then move onto the next one. This incremental approach will reduce your programming and your debugging time. Here is a suggestions for how to go about this problem:

1. **Read the board files:** Write a method capable of reading a board file and returning its contents as a 2D array of `int`. Test this method by calling it, and then passing its result to `Support.printAndVerifyInitialBoard()` to see if it worked correctly.

2. **Test legal placements:** Write a method that can determine whether the placement of a particular value in a particular position on the board is legal. That is, does placing a value v in row r and column c violate the rules because v already exists in r, in c, or in the sub-grid to which (r, c) belongs? Test this method by reading in a board file and then placing values that are known to be legal and illegal in particular positions, calling your tester method on each to see if it evaluates the situation correctly.

3. **Write the solver:** With an initial board loaded and a method that can determine whether the placement of a value into a blank location on the board is valid, you can now write the actual solver method. Use `Support.printResults()` to determine whether your solver is doing its job correctly.

Testing your code: There are three Sudoku board files:

- `easy.board`: The easiest of the puzzles to solve. Start with this one.
- `medium.board`: Fewer initial values are provided than in `easy.board`, making the search a bit harder.
- `evil.board`: A minimal number of values are provided. This solution requires backtracking to work correctly. (That is, certain values will be placed, but their incorrectness will not be evident until much later in the search.)

So long as you call the `Support` methods at the right times, you should be able simply to use these three provided puzzles to test and debug your code. Feel free, also, to find other puzzles and make your own Sudoku board files.

3 Submitting your work

Submit your `Sudoku.java` source code file with the CS submission system, using one of the two methods:

- **Web-based:** Visit the CS submission systems web page at `www.cs.amherst.edu/submit`.

- **Command-line based:** Use the `cssubmit` command at your shell prompt.
  (WARNING: This method works only on `remus/romulus`.)

This assignment is due on Tue, May-07, 11:59 pm.